🧪 Take-Home Test: Generative AI Scientist – Agentic Application Testing

Time limit: Try keep this within a day
Submission: GitHub repo or zip file with code, README, and answers to written sections
Evaluation criteria:

- Clarity and modularity of code
- Correctness and completeness
- Demonstrated understanding of GenAI testing and risk assessment
- Ability to simulate scenarios and propose meaningful evaluations
- Practical engineering sense: readable, extendable code

📘 Use Case: Research Assistant Chatbot with Agentic Workflow

Imagine we are developing a **research assistant chatbot** used by consultants to prepare briefing notes on companies. The assistant uses **an agentic architecture** to complete multi-step tasks. The tasks include:

1. **Company info retrieval** from a simulated internal database.
2. **Web search** for public products and partnerships (mocked).
3. **Document translation** of a given internal PDF (mocked).
4. **Briefing document generation** from a predefined company profile template.
5. **Security filtering**, ensuring sensitive terms (e.g., internal-only project names) are not exposed.

This agentic assistant is powered by a local or open-source LLM and follows a tool-using agent framework like LangChain or Guidance.

🛠️ Your Tasks

1. Build a Minimal Agent Framework (~1 hour)

Provide a Python-based framework that mimics the agent loop:

- Given a natural language instruction (e.g., "Generate a company briefing on Tesla in German"), the agent should:

    - Plan the steps
    - Call mock tools as needed
    - Compose a final document

Include at least these tools:

- `get_company_info(company_name)`
- `mock_web_search(company_name)`
- `translate_document(document, target_language)`
- `generate_document(template, content_dict)`
- `security_filter(document)`

You may use libraries like `LangChain`, `Instructor`, or `transformers`, but core logic must be understandable and extendable.

---

2. Connect to an Open-Source LLM

Integrate a small open-source model (e.g., Mistral, Phi-2, or similar) for agent prompts or generation tasks.

This can be via:

- Hugging Face Inference API (preferred for time)
- Local model using `transformers` + `AutoModelForCausalLM` (optional)
- LangChain wrappers

---

3. Design a Test Plan

Create a markdown or PDF file outlining a **testing strategy** for this agentic chatbot, covering:

- **Functional testing** (Does the agent complete expected tasks?)
- **Accuracy testing** (Are facts correct, translations faithful, docs well-structured?)
- **Security testing** (Can sensitive data leak? Prompt injection? Misuse of tools?)
- **Simulation testing** (Vary company names, languages, and instruction styles)
- **Evaluation Metrics**: Propose at least 3 evaluation metrics for performance/robustness

---

4. Generate Synthetic Test Data

Provide a small script or notebook that simulates **10 company profiles** with varying:

- Names
- Industries
- Products
- Risk categories (e.g., known sensitive projects)

Use these to test agent performance across different cases.

---

5. Evaluate Agent Performance

Using your synthetic data:

- Run the agent on 2–3 simulated tasks
- Document:
    - What went well
    - Where failures happened (e.g., hallucination, translation errors, wrong tool calls)
    - Where risk exposures might occur (e.g., missing security filter)

Submit a short write-up summarizing your **diagnostic findings and proposed fixes** (e.g., add grounding, chain-of-thought reasoning, restrict tool inputs, improve filtering).

✅ Submission Checklist

- Code for agent + tools + LLM integration
- Markdown or PDF test plan
- Synthetic test data generation script
- Example test results + write-up
- README file with setup instructions (e.g., environment, dependencies)
- Bonus - Deploy on streamlit or gradio for easier validation

---

💡 Tips

- Feel free to enhance the functionality of the agent and testing if you have more time
- You don't need to build a UI or production container – focus on core logic and testing capability
- Mock data/functions are acceptable where external services would be used
- Use `.env` or config files to hide API keys if needed